Izrada igre u labirintu pomoću alata Unity

Culović, Borneo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Borneo Culović

IZRADA IGRE U LABIRINTU POMOĆU ALATA UNITY

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Borneo Culović

Matični broj: 0246064592

Studij: Informacijski sustavi

IZRADA IGRE U LABIRINTU POMOĆU ALATA UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Robert Kudelić

Varaždin, lipanj 2021.

Borneo Culović

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sadržaj

1. Uvod

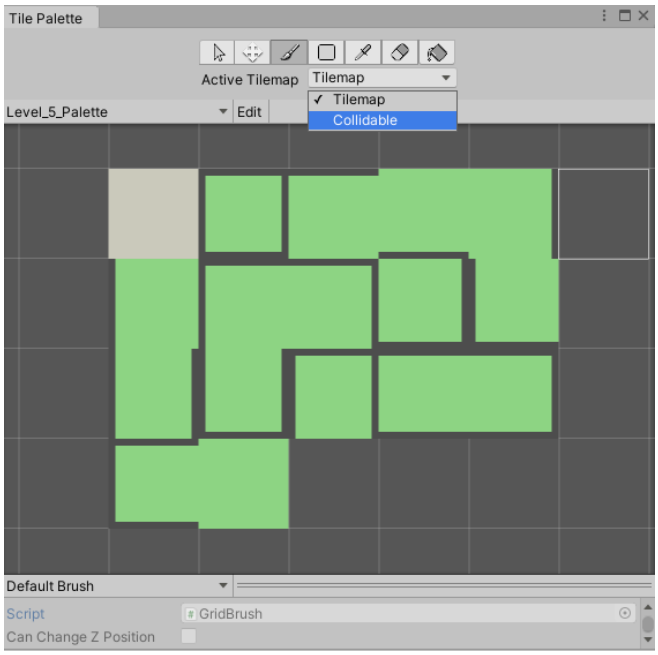
U ovom pismenom radu opisana je kreacija dvodimenzionalne računalne igre u programskom alatu Unity. Igra se sastoji od elemenata dviju postojećih računalnih igara, a to su „Pac-Man“ i „Bomberman“. Elementi „Pac-Mana“ koji se mogu vidjeti u ovoj igri su lovci koji love igrača dok skuplja novčiće po labirintu, a elementi „Bombermana“ koji se pronalaze u ovoj igri su mine koje igrač postavlja da bi eliminirao lovce. Kroz 5 razina igrač kontrolirajući svojeg pijuna kroz labirint mora sakupiti novčiće koje razmijeni za minu. Mine se nalaze na određenim lokacijama u labirintu te je potrebno imati 1500 bodova da se mina može pokupiti. Jednom kada igrač prikupi dovoljno bodova pokupi minu te ju postavlja iza sebe u svrhu eliminacije lovca koji ga proganja po labirintu. Na razini može biti i više aktivnih lovaca od jednom. Kada su svi lovci eliminarani glavna meta – „Ljigavac“, postaje ranjiv i prestane biti otporan i opasan po igrača. Ljigavac je smrtan za igrača ukoliko igrač dođe u fizički kontakt sa ljigavcem osim kada su svi lovci eliminirani. Igrač tada može samo projuriti kroz njega u znak pobjede. Svaki nivo je teži od predhodnog. Težina razine je određena brojem lovaca, brojem aktivnih lovaca, te količinom i veličinom prepreka u labirintu. Postoji jednostavan način za otarasit se lovca koji je neposredno blizu igrača, a to je da se pronađe prepreka oko koje se mogu raditi krugovi. Razlog zašto se lagano otarasiti lovca kretanjem u krug je taj što se lovac ne može savršeno kretati niti kratiti uglove, te ga inercija zanese pri svakom skretanju. Na višim razinama je teže pronaći dovoljno velike prepreke za kruženje, te pošto su dva aktivna lovca na kasnijim razinama to otežava bilokakvo kretanje u krug pošto učestalo drugi lovac (koji je dalje od igrača) krene u krug u suprotnu stranu. Kao dugogodišnji entuzijast računalnih igara imao sam u naumu napraviti igru koja zahtjeva što kraće vrijeme donošenja odluke. Stoga igra opisana ovim radom je igra brze odluke i snalaženja u labirintu.

2. Metode i tehnike rada

2.1. Općenito o korištenim metodama i tehnikama

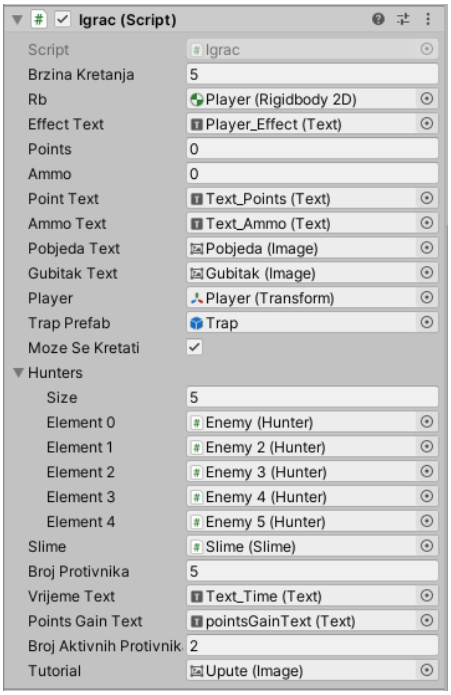
Pri izradi teme, točnije, računalne igre korišten je računalni alat Unity [1] (korištena verzija je 2019.3.15f1). To je platforma za stvaranje interaktivnog sadržaja u stvarnom vremenu [2]. Sadrži sve potrebne elemente i funkcije za kreaciju programske logike i okoline jedne računalne igre. Osim toga kao što su i J. Craighead, J. Burke i R. Murphy zamijetili [3], Unity dolazi sa kompletnom dokumentacijom i primjerima za razliku od nekih drugih sličnih alata. U ovom radu se radi o dvodimenzionalnoj igri stoga su korištene neke metode izrade određenih dijelova igre specifične za razvoj dvodimenzionalne igre.

Za kreaciju labirinta korištena je opcija „tilemap“ u Unity alatu. To je metoda kojom se razina može graditi „pločicama“. Prije toga je bilo potrebno u jednom od alata za uređivanje ili kreiranje fotografija oslikati sve elemente potrebne za gradnju labirinta, primjerice hodnik koji je ograđen zidom sa jedne strane, hodnik ograđen zidom sa više strana, blok prepreke, blok po kojem će se igrač moći kretati i sl. Fotografije koje sadrže elemente za izgradnju labirinta su izrezane na kvadratiće istih dimenzija, odnosno „pločice“, unutar Unity alata. Taj skup pločica se naziva „tileset“ ili skup pločica. Nakon toga pošto se koristi „tilemap“ metoda gradnje razine potrebno je samo odabrati vrstu pločice sa palete koju želimo naslikati na podlogu i nanijeti ju kao kistom. Pločice se mogu oslikavati u više razina. Na slici 1 možemo vidjeti da postoje dva sloja, a to su „tilemap“ i „collidable“. Razlog zašto su dva sloja je taj što se odabirom jednog sloja („tilemap“) oslikavaju polja kojima igrač i protivnici mogu kročiti, a odabirom drugog sloja se oslikavaju pločice koje predstavljaju prepreke ili zidove.



Slika 1: Paleta pločica (autorova slika 2021)

Unity sam po sebi ne služi za stvaranje programske logike ili opisivanje događaja koji nastanu zbog određenih uvijeta. Za te svrhe je potreban programski alat za pisanje koda. U ovom radu korišten je Visual Studio 2019 [4]. Programska logika računalne igre izrađene u Unity alatu se stvara u takozvanim skriptama u C# programskom jeziku. Skripta se otvori u alatu za pisanje koda te se dodaju varijable i funkcije. Te skripte se mogu stvoriti direktno iz Unity projekta ili zasebno pa se umetnu u projekt. Jednom kada se skripta kreira moguće ju je primjeniti na bilo koji od objekta koji su kreirani na trenutnoj razini. Tada se odabirom objekta kojega smo povezali sa skriptom pojavljuje odjeljak sa varijablama iz skripte (slika 2).



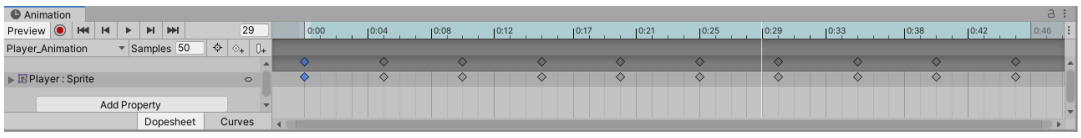
Slika 2: Unity prikaz skripte i njezinih varijabli (autorova slika 2021)

U tom dijelu sučelja se upisuju vrijednosti koje se tokom igre proslijede skripti da bi skripta znala što i sa čime treba raditi te time vraća vrijednosti ili neke podatke Unity alatu na izvršavanje promijena kao što su primjerice lokacija na kojoj se nalazi igrač, lovac ili jednostavno dodavanje bodova nakon pokupljenog novčića. Svaka skripta ima 3 tipa funkcije. Jedna je funkcija koja se pokrene pri samom pokretanju skripte, funkcija koja se pokreće kontinuirano za svaku promijenu sličice na ekranu i funkcije koje se pokreću na zahtijev Unity alata. Funkcija koja se pokrene na početku je „Start()“ funkcija te je u njoj najbolje napraviti inicijalizacije [5]. „Update()“ ili „FixedUpdate()“ su funkcije koje se pokreću za svaku sličicu stoga se unutar tih funkcija stavlja logika koja se izvodi gotovo konstantno (npr. kretanje). Ostale funkcije se okidaju na poziv.

Unity omogućava kreiranje više scena koje se koriste u igri. Jedna scena je, primjerice, glavni izbornik u kojem igrač odabire želi li započeti novu igru ili izaći iz igre. Jedna scena može biti jedna od razina. Unity omogućava kreaciju i opis scena, te uz skripte se postiže smislena navigacija između njih.

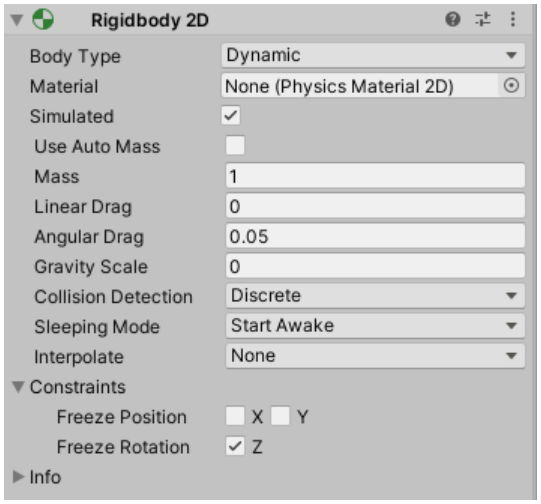
Zvuk korišten u ovoj igri stvaran je lupkanjem i dodirivanjem objekata, te proizvedenim zvukovima vlastog glasa. Za snimanje zvuka korišten je alat za uređivanje i snimanje zvuka – Audacity [6]. Zvukovni zapisi dodani su u projekt unutar Unitya te je kreirana skripta koja rukuje sa zapisima i prepoznaje kada se koji zapis treba pustiti.

U izradi pijuna kojime igrač upravlja, lovca i ljigavca koristi se mogućnost animacije. Korištenjem animacije postignute su dvije stvari, a to su dinamični izgled igre i slanje određenih vizualnih informacija igraču. Primjer takvih vizualnih informacija je ljigavac koji se povremeno može okameniti. Kada je aktivan i kada se kreće zelene je boje te se njegova tri oka otvaraju i zatvaraju periodički, a kada je neaktivan (okamenjen) onda je sive boje i ukipljen. Animacije su napravljene mjenjanjem niza sličica određenom brzinom. Dakle izgled svake zasebne sličice koja se pušta u animativnom nizu opisan je u alatu za uređivanju slika te kada se te sličice puste u određenom rasporedu i uskladi brzina izmijene tih sličica dobiva se animacija. Samo je potrebno dodijeliti animaciju objektu za kojeg želimo da se animira.

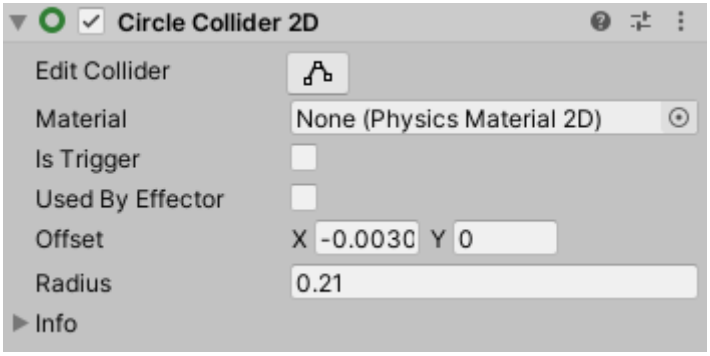


Slika 3: Kreiranje animacije u Unityju (autorova slika 2021)

Ponašanje objekata s obzirom na zakone fizike je postignuto uz pomoć predefiniranih skripti u Unity alatu. Dvije predefinirane skripte su skripte sudarača i krutog tijela (eng. *Collider and Rigidbody*). Navedene skripte imaju verzije za dvodimenzionalne i trodimenzionalne verzije igre. U ovoj igri su korištene dvodimenzionalne verzije.



Slika 4: Skripta krutog tijela (autorova slika 2021)



Slika 5: Skripta sudarača (autorova slika 2021)

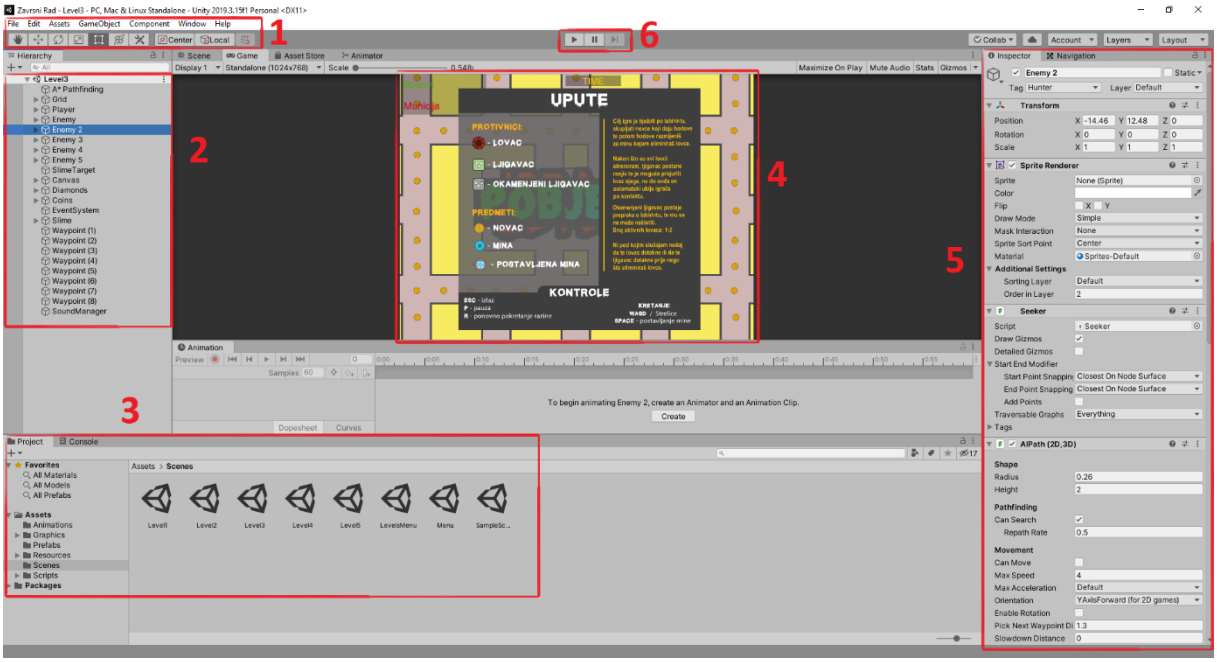
Skripta krutog tijela sastoji se od nekoliko varijabli koje definiraju kako će se objekt ponašati jednom kada se igra započne. Ono što je ključno za zamijetiti je varijabla „Gravity scale“ koja je postavljena na vrijednost nula. Ukoliko se ta varijabla postavi na vrijednost 1 tada svi objekti automatski padaju dole jer se ponašaju kao da ih gravitacija privlači. U ovoj igri na svim pojavljivanjima te skripte je ta varijabla postavljena na vrijednost nule jer je željeno postići osjećaj da se igra promatra odozgo.

Sudarači su skripte koje rade rub oko objekta kojemu su pridružene i taj rub kada dođe u doticaj sa drugim objektom spriječi prolaženje objekta kroz dodirnuti objekt. Ukoliko se skripte sudarača u potpunosti maknu iz igre tada bi svi objekti mogli prolaziti kroz sve ostale objekte. Skriptom sudarača se postiže kreiranje „zidova“ i stvara se dojam da su određeni objekti kruta tvar koja se može sudarati sa zidovima ili drugim objektima. Sudarači dolaze u raznim oblicima

a najkorišteniji u ovom radu su „circle“ i „rectangle“ odnosno kružnica i kvadrat. Na postavkama skripte sa slike 5, potrebno je izdvojiti varijablu „Is Trigger“ što u prijevodu znači da ako se označi ta varijabla objekt postaje okidač. Ukoliko je neki objekt okidač tada se ne može sudarati sa drugim objektima, ali se pri kontaktu sa drugim objektom okida određena programska logika.

2.2. Unity sučelje

U ovom će se podpoglavlju govoriti o sučelju i mogućnostima koje nude pojedini dijelovi tog sučelja. Ti pojedini dijelovi sučelja su ujedno i oni koji su se najviše koristili u razvoju ove igre.



Slika 6: Unity sučelje (autorova slika 2021)

1) **Alatna traka** sadrži neke najosnovnije funkcije alata. U gornjem dijelu nalaze se standardne opcije kao i u mnogim drugim računalnim alatima, no ono što je više specifično za Unity je donja traka u kojoj se nalaze alati za manipulaciju objektima koji se nalaze u igri. Recimo alat za pomicanje odabranog objekta, povećavanje/smanjivanje, rotacija i sl.

2) **Hijerarhija objekata** jest lista objekata koji se nalaze u sceni. Naziv trenutačno odabrane scene se nalazi pri samom vrhu označenog kvadrata te sve ispod toga su objekti na toj sceni. Objekti se mogu smještati jedni u druge što utječe na krajnje ponašanje tih objekata u toj sceni.

3) **Projekt** je odjeljak označen brojem tri unutar kojega možemo vidjeti sve datoteke priložene u projektu. To su sve slike, zvukovi, palete „pločica“ (Eng. *Tileset palette*), animacije, scene, ali i već kreirani objekti koje ne trebamo nužno imati u igri (Eng. *Prefab*). Pored odjeljka sa datotekama koje čine projekt nalazi se konzolni odjeljak koji kada se odabere prikaže nam se polje u koje se ispisuju poruke igre. To mogu biti poruke sa detaljima o greškama koje se događaju u radu igre ili poruke koje sami zadamo da se ispišu u nekoj od skripata.

4) **Igra** ili, više točnije, ono što glavna kamera vidi u trenutku kada se pokrene igra. Taj dio je ustvari pregled postavljenih objekata u scenu. Odabirom objekta sa hijerarhije naznači se taj objekt u sceni te ga potom možemo oblikovati ili pomicati sa jednim od alata sa alatne trake.

5) **Prozor sa varijablama i postavkama** je nazvan „Inspector“ na slici te na toj desnoj strani sučelja vidimo sve što se može vidjeti o odabranom objektu. Ukoliko niti jedan objekt nije odabran sa hijerarhije objekata tada je taj dio prazan. Iako su svi dijelovi sučelja bitni ovo je dio u kojemu se rade najveće izmijene. U njemu se dodaju razne funkcije ili ponašanja odabranom objektu, mijenja fizika ponašanja objekata tokom igre, pristupa se vrijednostima varijabli definiranima u skriptama (ukoliko je skripta dodijeljena odabranom objektu) i mnoge druge mogućnosti.

6) **Kontrolne tipke** se nalaze na alatnoj traci te se sastoje od tri tipke. Lijeva pokreće igru, srednja ju pauzira, a desna ju zaustavlja u potpunosti.

3. Opis izrade igre

U ovom poglavlju biti će detaljno razrađene metode i načini kojima su kreirane funkcionalnosti igre. Prvo će biti predstavljene scene te veze između njih, potom opis skripti koje se koriste, opis zvukova koji su korišteni u igri i na kraju razrada kako mine funkcioniraju.

3.1. Scene

U ovom projektu postoji sveukupno 7 scena. Pet od njih predstavljaju razine igre, a dvije predstavljaju izbornike. Prva scena se u projektu naziva „Menu“ i ona je glavni izbornik ove igre. Ima dvije opcije a to su opcije za izlaz iz igre i za započimanje nove igre.

Slika 7: Glavni izbornik (autorova slika 2021)

Pritiskom na tipku za izlaz igra se automatski gasi, a pritiskom na tipku „Nova igra“ pojavljuje se scena imenom „LevelsMenu“ u kojoj se nalazi 6 opcija od kojih su 5 tipke za razine (1 - 5) te tipka „Izlaz“ za povratak na glavni izbornik.

Slika 8: Izbornik sa razinama (autorova slika 2021)

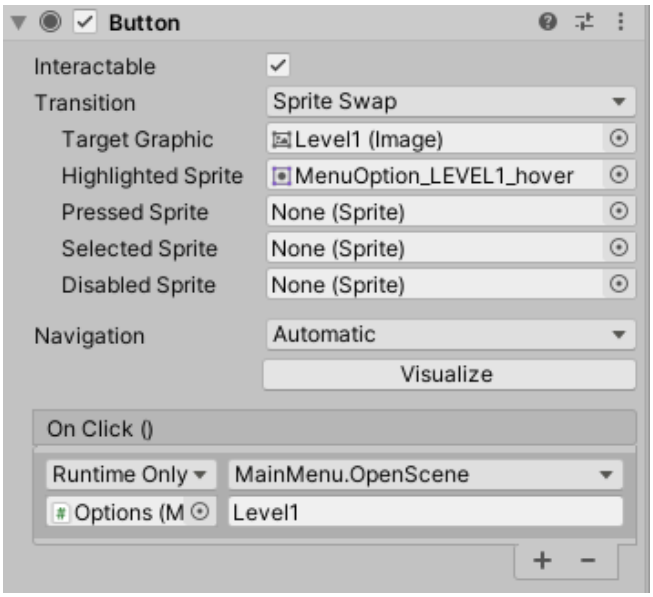
Mijenjanje scena implementirano je pomoću već postojeće klase „SceneManager“. Ta klasa sadrži metode za rad sa scenama te se u ovoj igri najviše koristi funkcija „LoadScene()“. No osim klase „SceneManager“ koristi se i kreirana skripta koja obuhvaća sveukupni rad izbornika, a zove se „MainMenu“ i izgleda ovako:

```
public class MainMenu : MonoBehaviour
{
    public void OpenScene(string scene)
    {
        SceneManager.LoadScene(scene);
    }

    public void QuitGame()
    {
        Application.Quit();
    }

}
```

Potom se na sve tipke pod „OnClick()“ funkcijom namijesti da se pokrene funkcija „OpenScene()“ sa prikazanog koda te se proslijedi ime scene koja se želi prikazati nakon pritiska na tipku.



Slika 9: Postavke tipke za prebacivanje na scenu „Level1“ (autorova slika 2021)

Varijable „Target Graphic“ i „Highlighted Sprite“ su varijable kojima se pridružuju slike kako želimo da tipka izgleda i kako želimo da izgleda kada se pokazivačem prijeđe preko tipke. Na dnu slike 9, nalazi se dio bitan za promijenu scene. Na toj slici je primjer kako to izgleda u postavkama tipke koja preusmjerava na scenu „Level1“. Gornji lijevi padajući izbornik u odjeljku „On Click()“ funkcije određuje kada će se registrirati interakcija sa tipkom. Može se isključiti potpuno, uključiti samo kada se igra pokrene ili čak i kada igra nije pokrenuta nego je projekt samo otvoren. Do toga padajućeg izbornika nalazi se drugi padajući izbornik koji sadrži sve funkcije i varijable sa skripte „MainMenu“ te je u ovom slučaju odabrana „OpenScene()“ funkcija jer je to funkcionalnost koja je željena na tipki za promijenu scene. Donji dio se sastoji od varijable koja se proslijeđuje skripti i u ovom slučaju je to vrijednost „Level1“ što je ujedno i ime scene koju želimo vidjeti. Na isti način su implementirane sve tipke u projektu.

Uz pomoć spomenutih funkcija je implementirano ponovno pokretanje razine. Odnosno kada igrač umre ili poželi krenuti isponova pritiskom slova „R“ sa tipkovnice pokreće funkciju „LoadScene()“ iz klase „SceneManager“ te joj se za parametar proslijedi ime trenutačno aktivne scene uz pomoć poziva funkcije GetActiveScene() te njezine varijable „name“.

3.2. Skripte

U ovom se radu nalazi 8 skripti. Od kojih je jedna već opisana i ta je povezana sa navigacijom po izborniku i scenama, pet skripti koje implementiraju preostale funkcionalnosti igre, dvije koje implementiraju zvuk, te sktipte za traženje puta do cilja sa preprekama (A* algoritam).

3.2.1. Igrac.cs

Ova skripta sadrži mnoge funkcionalnosti u koje spada kretanje igrača, izmijene aktivnih lovaca, puštanje zvukova, dodavanje bodova, registriranje kada je kraj igre, veliki dio korisničkog sučelja koje je prikazano tokom igre, te rukuje i sa sudarima između objekata i igrača.

U skriptu su dodane reference na biblioteke UnityEngine-a „UI“ i, već spomenuta, „SceneManager“. „UI“ biblioteka sadrži potrebne metode za rad sa slikama i tekstom te omogućuje stvaranje korisničkog sučelja za prikaz informacija tokom igre. Primjerice broj ostvarenih bodova, broj mina koje igrač sadržava, vrijeme proteklo od početka i sl. Na samom se početku u funkciji „Start()“ izvršavaju naredbe koje postavljaju korisničko sučelje, zaustavljaju igru u svrhu prikaza uputa, skrivaju neke elemente koji se nebi trebali prikazivati na samom početku, te pozivaju funkciju koja se periodički poziva, a to je funkcija za promijenom aktivnog lovca. U funkciji „Update()“ se izvršava niz provjera, a jedna od najvažnijih za napomenuti su provjere određenih tipki jesu li pritisnute na tipkovnici. Te konstantne provjere omogućavaju igraču da kontrolira svog pijuna, privremeno zaustavi igru, započne nivo isponova ili izađe van iz igre. Ukoliko igrač pritisne razmaknicu kada je igra zaustavljena i upute su prikazane na ekranu, tada se upute skrivaju i igra nastavlja, ukoliko upute nisu uključene tada ta tipka vrši funkciju postavljanja mine. Tipka „R“ započinje igru isponova, odnosno pokreće trenutačnu scenu isponova. Slovo „P“ privremeno zaustavlja igru, odnosno – pauzira ju, te tipka „ESC“ odnosno „escape“ izlazi iz igre van u izbornik sa razinama. Ovo je algoritam koji radi opisane provjere.

```
kretnja.x = Input.GetAxisRaw("Horizontal");
kretnja.y = Input.GetAxisRaw("Vertical");
if(Input.GetKeyDown(KeyCode.Space) && tutorialActive)
{
    tutorialActive = false;
    tutorial.enabled = false;
    Time.timeScale = 1;
    InvokeRepeating("DodajBodove", 10f, 10f);
}
if (Input.GetKeyDown(KeyCode.Space) && Ammo > 0)
{
    Ammo--;
    ammoText.text = "Municija: " + Ammo.ToString();
    Instantiate(trapPrefab, player.position, player.rotation);
    FindObjectOfType<SoundManager>().Play("mineDrop");
}
```

```
if (Input.GetKeyDown(KeyCode.R))
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
if (Input.GetKeyDown(KeyCode.P))
{
    if(Time.timeScale == 0)
    {
        Time.timeScale = 1;
    } else
    {
        Time.timeScale = 0;
    }
}
if (Input.GetKeyDown(KeyCode.Escape))
{
    if (tutorialActive)
    {
        tutorialActive = false;
        tutorial.enabled = false;
        Time.timeScale = 1;
        InvokeRepeating("DodajBodove", 10f, 10f);
    }
    else
    {
        SceneManager.LoadScene("LevelsMenu");
    }

}
```

Osim tih provjera u „Update()“ funkciji obavlja se i brojanje sekundi. To je potrebno raditi zbog toga što se svakih par sekundi mijenjaju aktivni lovci i dodaju bodovi. Bodovi se dodaju svakih 10 sekundi što je igrač živ, te dobiva 10 bodova za svakog živog lovca u labirintu. Što znači da ako su 3 živa lovca i ako je dotaknuta vremenska crta nakon koje se odbrojavanje vremena resetira, tada igrač dobije 30 bodova.

U „FixedUpdate()“ funkciji se odvija kretanje pijuna ovisno o pritisnutim tipkama (strijelice ili „WASD“ slova) i duljini vremena što su tipke pritisnute. Osim navedenih funkcija u ovoj se skripti još isplati spomenuti skripta za promijenom aktivnog lovca. Funkcija se zove „PromijeniHuntera()“ i poziva se svakih par sekundi. U toj se funkciji prvo obavi provjera broja

lovaca jer ako je samo jedan ili nijedan onda nema smisla izvršavati naredbe i ostale provjere. Ukoliko ih ima više od jednoga tada se svim lovcima sa liste oduzima mogućnost kretanja. Potom „n“ broj puta (gdje je „n“ broj koji predstavlja količinu aktivnih lovaca na nivou) izvrši funkcija „Random()“ koja je predefinirana u „UnityEngine“ biblioteci te vraća neki nasumični broj u određenom intervalu. U ovom slučaju taj interval je od 0 do jedan manje od koliko god lovaca je preostalo na razini. Dakle ako su 4 lovca tada je to interval od 0 do 3. Te se za svaku pojedinu iteraciju ponavlja dobivanje nasumičnog broja dok lovac pod tim rednim brojem u listi ne bude adekvatan odabir, a da bi on to bio sve što je potrebno je da je još uvijek živ. Tada mu se dodijeli pravo kretanja i on tako kreće u lov. Potom kada se „n“ lovaca aktivira (gdje je „n“ maksimalni broj aktivnih lovaca) tada se određuje vrijeme njihove/njegove aktivnosti na isti način, odnosno, pozove se funkcija „Random()“ samo ovaj put u intervalu od 9 do 16. No ukoliko je samo 1 aktivan lovac tada se, preventivno, svim živim lovcima daje pravo kretanja i na isti način se postavlja vrijeme koje će ti lovci biti aktivni. Kada prođe taj zadani vremenski interval cijela procedura se ponavlja. Cijeli algoritam te logike izgleda ovako:

```
public void PromijeniHuntera()
    {
        if(hunters.Count > 1 && hunters.Count >= BrojAktivnihProtivnika)
        {
            foreach (Hunter h in hunters)
            {
                if (h.astarAIbase.canMove)
                {
                    h.astarAIbase.canMove = false;
                }
            }
            for (int x = 1; x <= BrojAktivnihProtivnika; x++)
            {
                do
                {
                    randnum = UnityEngine.Random.Range(0, brojProtivnika);
                } while (hunters[randnum].Mrtav ||
hunters[randnum].astarAIbase.canMove);
                hunters[randnum].astarAIbase.canMove = true;
                Debug.Log(hunters[randnum].name);
            }
            timeLeft = UnityEngine.Random.Range(9f, 16f);
            FindObjectOfType<SoundManager>().Play("hunterSwitch");
        } else
```

```
    {
        foreach (Hunter h in hunters)
        {
            if (!h.astarAIbase.canMove)
            {
                h.astarAIbase.canMove = true;
            }
        }
        timeLeft = UnityEngine.Random.Range(9f, 16f);
    }
}
```

U istoj se skripti nalaze i funkcije koje određuju što će se dogoditi kada se igrač sudari sa drugim elementom u nivou. To se sve izvršava kroz dvije funkcije „OnTriggerEnter2D“ i „OnCollisionEnter2D“. Razlika između njih je ta što kod prve funkcije jedan od objekata je u svojoj skripti sudarača imao uključenu opciju da je okidač, a to nije igrač, što znači da se sudar neće registrirati u fizičkom smislu, ali će se pokrenuti određena logika, a ta logika je provjera oznake objekta sa kojim se igrač sudario, odnosno kroz koji okidač je prošao. Postoje 3 stvari koje se provjeravaju kada igrač prođe kroz okidač. Prvi objekt koji može biti je novčić, te u tom slučaju igrač dobiva 10 bodova, osvježava mu se korisničko sučelje, novčić se briše i čuje se zvuk skupljanja novčića. Drugi objekt je dijamant, odnosno mina koju igrač pokupi ukoliko ima 1500 bodova ili više. Ukoliko igrač ima preko 1500 bodova tada se objekt dijamanta briše, igraču se dodaje mina u spremnik, igrač gubi 450 bodova, osvježava se korisničko sučelje i pušta se zvuk skupljanja mine. Treća opcija je ljigavac. Ukoliko su svi lovci mrtvi, ljigavac se briše i igrača se proglašava pobjednikom, no ukoliko nisu tada igrač umire i postaje gubitnik. Druga funkcija obrađuje sudare sa lovcima nakon kojega igrač postaje gubitnik. U toj funkciji postoji logika koja se bavi sudarima sa ljigavcem no kako je ljigavčev sudarač ustvari okidač, tada taj dio koda nema svrhu, ali otvara vrata budućim nadogradnjama. Ovako izgleda algoritam za obrađivanje sudara:

```
private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Coins") && mozeSeKretati)
        {
            Points+= 10;
            pointText.text = "Bodovi: " + Points.ToString();
            Destroy(collision.gameObject);
            FindObjectOfType<SoundManager>().Play("coin");
        }

        if       (collision.gameObject.CompareTag("Diamond")       &&
mozeSeKretati)
        {
```

```
                if(Points >= 1500)
                {
                    Ammo++;
                    ammoText.text = "Municija: " + Ammo.ToString();
                    Points -= 450;
                    pointText.text = "Bodovi: " + Points.ToString();
                    Destroy(collision.gameObject);
                    FindObjectOfType<SoundManager>().Play("minePickUp");
                }
            }
            if (collision.gameObject.CompareTag("Slime"))
            {
                if (slime.MozeUmrijeti)
                {
                    Destroy(collision.gameObject);
                    slime.Ziv = false;
                    Pobjeda();
                }
                else if (slime.MozeSeKretati == true)
                {
                    Gubitak();
                }
            }
    }
```

Ovaj algoritam služi za obradu okidača:

```
private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Hunter") && Ziv)
        {
            Gubitak();
        }

        if (collision.gameObject.CompareTag("Slime"))
        {
            if (slime.MozeUmrijeti)
            {
                Destroy(collision.gameObject);
                Pobjeda();
            } else if(slime.MozeSeKretati == true && Ziv)
            {
                Gubitak();
            }
        }
    }
```

Za kraj u toj skripti postoje još dvije kratke funkcije, a to su „Pobjeda()“ i „Gubitak()“ koje isključuju mogućnost kretanja igrača te na sedam sekundi prikažu ispravnu sliku ovisno radi li se o pobjedi ili gubitku, te nakon isteka tih sedam sekundi „Update()“ funkcija vrati igrača u izbornik sa razinama.

3.2.2. Slime.cs

Ova skripta je pridružena ljigavcu te određuje ponašanje i kretanje ljigavca. Sadrži varijable koje služe za određivanje stanja ljigavca ili za dodavanje, odnosno oduzimanje, mehanika sa razine. Na početku u „Start()“ funkciji se postavlja početna točka do koje će ljigavac pokušati doći i prikazuju se dvije slike koje izgledaju kao jedna, a to je lubanja na podlozi koja označava igraču da je ljigavac smrtonosan na dodir.

U „Update()“ funkciji se regulira brzina kretanja ljigavca. Ljigavac se u početku kreće jako brzo te onda do određene brzine usporava ovisno o bodovima koje igrač ima. Također osim regulacije brzine se vrši provjera jesu li svi lovci uništeni, te ako jesu postavlja zastavicu „MozeUmrijeti“ na istinu što gasi opciju da se ljigavac okameni povremeno. Hoće li se periodički ljigavac okameniti ili neće ovisi o varijabli „StoneMechanism“. Ukoliko je vrijednost te varijable „true“ tada se svakih 10 sekundi mijenja stanje ljigavca, no ako je vrijednost „false“ tada se ljigavac ne okamenjuje već se slobodno može kretati.

Postoje 2 načina određivanja kretanja. Jedan od načina je taj da se u varijablu „waypoints“ proslijede točke iz labirinta. Tada ljigavac pokušava redom ići od prve do posljednje točke sa liste, te ukoliko dođe do zadnje tada se kreće prema prvoj sa liste i tako u krug. Ono što je bitno za ovakav način navođenja je to da između dvije slijedne točke ne postoji prepreka jer se ljigavac neće moći kretati. Takav način navođenja ljigavca je najbolje koristiti kada je bitno da ljigavac prođe ili prolazi određenim dijelom labirinta. Drugi način navođenja uključuje A* algoritam, odnosno algoritam za traženje puta. Taj drugi način navođenja će biti objašnjen u idućoj cjelini pošto se njegova logika nalazi u drugoj skripti.

3.2.3. Waypoint.cs

Skripta koja ima samo jednu funkcionalnost, a to je podrška drugom načinu navigiranja ljigavca. Na ljigavca je primjenjen algoritam za traženje puta i za metu mu je postavljena točka kojoj se doda ova skripta. Ova skripta prima samo 2 parametra i to je ljigavac i lista pozicija, odnosno, ista onakva lista kao i za prvi način samo što kod ove liste dvije uzastopne točke mogu imati i prepreke između. U „Update()“ funkciji ove skripte se provjerava postoji li ljigavac te ako postoji provjerava se udaljenost do točke do koje ljigavac

pokušava doći. Ukoliko je ljigavac dovoljno blizu točki, okida se funkcija „NextPosition()“ koja točki mjenja poziciju. Nova se pozicija točke određuje na način da se generira nasumični broj od nule do jedan manje od sveukupnog broja mogućih pozicija i iz liste pod tim indexom se učita pozicija i dodijeli točki do koje ljigavac pokušava doći. Na taj način se kretanje ljigavca ne može predvidjeti, odnosno ne postoji obrazac po kojemu se ljigavac kreće. Ovako izgledaju „Update()“ i „NextPosition()“ funkcije:

```
public Transform[] positions;
    public Slime slime;
    void Start()
    {

    }

    void Update()
    {
        if(slime != null)
        {
            if (Vector2.Distance(slime.transform.position,
gameObject.transform.position) < 0.5f)
            {
                gameObject.transform.position = NextPosition();
            }
        }
    }

    private Vector2 NextPosition()
    {
        int randNum = Random.Range(0, positions.Length - 1);
        return positions[randNum].transform.position;
    }
```

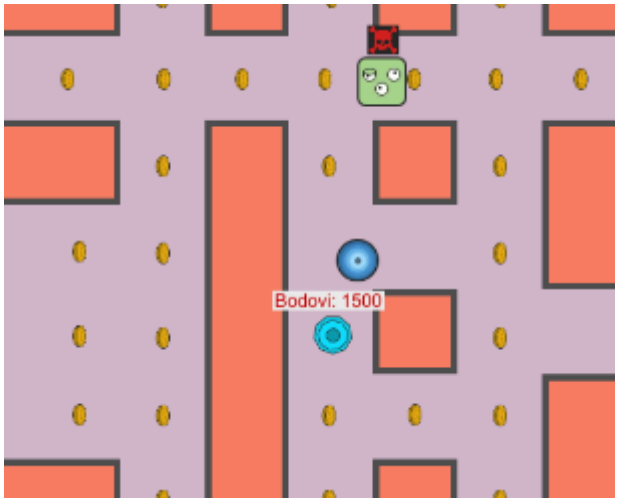
3.2.4. Hunter.cs

Ova se skripta dodaje svim lovcima na razini. Ona je zaslužna za vizualno informiranje igrača kada je lovac neaktivan, te uništavanje lovaca kada dođu u kontakt sa minom. U „Start()“ funkciji se ne dešava puno toga osim što se mali natpis „NEAKTIVAN“, koji je uobičajeno vidljiv kada je lovac zamrznut ili neaktivan, sakrije. U „Update()“ funkciji se provjerava nalazi li se lovac blizu drugih neaktivnih lovaca, a ta se provjera izvodi iz razloga što ako se lovac nalazi u blizini jednog neaktivnog lovca tada postaje brži te mu je brzina

povećana dok se god nalazi u radijusu od neaktivnog lovca. Tom mehanikom se stvara pritisak na igrača pošto je primoran napraviti nekoliko skretanja i ujedno pokušati natjerati lovca da izađe van iz tog radijusa. Također se vrši i provjera može li se lovac kretati te ako se ne može prikazuje se, već spomenuti, natpis „NEAKTIVAN“. Logika brisanja lovca prilikom kontakta sa minom opisana je u predefiniranoj funkciji „OnTriggerEnter2D“ koja se okida kada lovac dođe u kontakt sa minom čiji sudarač je okidač.

3.2.5. Diamond.cs

Ova skripta je veoma kratka i služi primarno tome da se igraču signalizira ukoliko dijamant kojemu prilazi može biti pokupljen, odnosno ima li igrač dovoljno bodova da ih zamijeni za jedan dijamant (minu). Vizualni signal igraču izgleda u obliku teksta koji se pojavi iznad dijamanta na kojemu piše koliko bodova je potrebno imati da sa mina može pokupiti. Primjer takve situacije nalazi se na slici 15.



Slika 10: Vizualni signal igraču da mu fali bodova (autorova slika 2021)

Skripta u „Update()“ funkciji provjerava nalazi li se igrač u određenom radijusu i ima li 1500 bodova što je fiksirana vrijednost potrebna da igrač bude u mogućnosti pokupiti dijamant, te ukoliko nema 1500 bodova tada se natpis prikazuje.

3.2.6. SoundManager.cs i Sound.cs

SoundManager.cs je skripta dodana u nevidljivi element unutar svake razine koja prima polje zvukova kao parametar. Štoviše polje tih zvukova je polje podataka tipa „Sound“

što je klasa kreirana radi lakšeg poziva zvukova dodanih u projekt. Definicija klase „Sound“ izgleda ovako:

```
[System.Serializable]
public class Sound
{
    public string name;
    public AudioClip clip;

    [Range(0f, 1f)]
    public float volume;

    [HideInInspector]
    public AudioSource source;
}
```

SoundManager.cs skripta ne sadrži „Start()“ funkciju. Umjesto nje sadrži „Awake()“ funkciju što je slična stvar osim što se „Awake()“ pokreće prije „Start()“ funkcije. Cijeli dio sa zvukom u igri napravljen je po uputama programera, koji na platformi YouTube objavljuje razne edukacijske video sadržaje vezane za Unity, Brackeysa [7]. SoundManager klasa (skripta) sadrži bitnu funkciju „Play“ koja za parametar prima ime. Ta funkcija pretražuje listu „Sound“ tipa podataka po imenu te pušta zatraženi zvuk. Algoritam spomenute funkcije izgleda ovako:

```
public void Play (string name)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        s.source.Play();
    }
```

3.2.7. Algoritam za traženje puta 'A*'

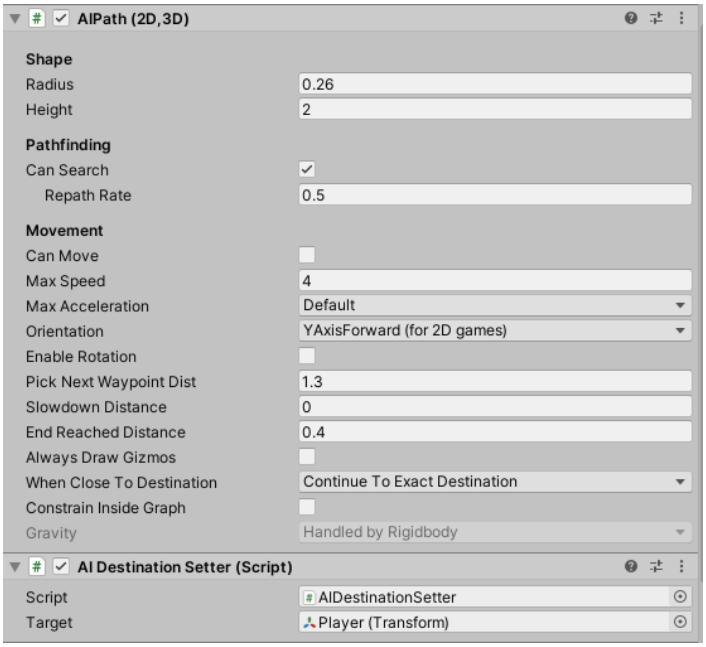
Algoritam za traženje puta, ili A* algoritam (izgovor: „A-star algoritam“) je algoritam koji se primjenjuje na lovce i ljigavca da bi na najbrži način došli do cilja zaobilazeći prepreke. Taj algoritam je preuzet sa interneta [8] i dodan u projekt te je jako prikladan labirintima u ovoj igri s obzirom da su labirinti građeni pločicama. Algoritam je skladan labirintu napravljenim od pločica (eng. *Tilemap*) zbog toga što algoritam nudi opciju za označavanje podloge koju treba skenirati, prepozna sa skeniranog područja pločice napravljene u određenom sloju te ih registrira kao prepreke. Zato je potrebno pri izgradnji labirinta paziti na slojeve da zidovi i nepristupačni dijelovi budu u sloju koji se kasnije pri osposobljavanju A* algoritma prepozna kao nepristupačan dio. Unutar biblioteke sa algoritmom nalazi se nekoliko klasa koje se mogu izmijeniti pa se primjerice lovac ne može kretati ili se kreće određenom brzinom. Isključujući i uključujući neke od tih osnovnih logika je nastala mehanika ove igre. Kada je u labirintu 5 lovaca, a samo su dva aktivna tada je kod onih neaktivnih unutar algoritma za navođenje isključena mogućnost kretanja. Za početak se na svaku razinu doda A* algoritam te se u njegovim postavkama unesu potrebni podaci i vrijednosti koje se odnose na dimenzije jednog polja (pločice). Pritiskom na tipku scan, algoritam prepozna koja polja su prepreke, a po kojim se poljima može kretati.

Slika 11: Konfiguracija A* algoritma (autorova slika 2021)

Potom je potrebno svakom zasebnom objektu za kojeg želimo da koristi algoritam za navođenje kroz labirint pridružiti tri skripte iz biblioteke algoritma, a to su:

- Seeker
- AIPath (2D,3D)
- AI Destination Setter

AIPath skripta je već spomenuta skripta koja sadrži varijable i zastavice o tome može li se objekt kretati i kojom brzinom će se kretati. Ta skripta je učestalo manipulirana u ovoj igri, primjerice varijabla „Can Move“ sa slike 19. AI Destination Setter je skripta kojoj se prosljeđuje objekt do kojeg želimo da se objekt na kojeg je primijenjen algoritam kreće.



Slika 12: AIPath i AI Destination Setter skripte (autorova slika 2021)

3.3. Zvuk

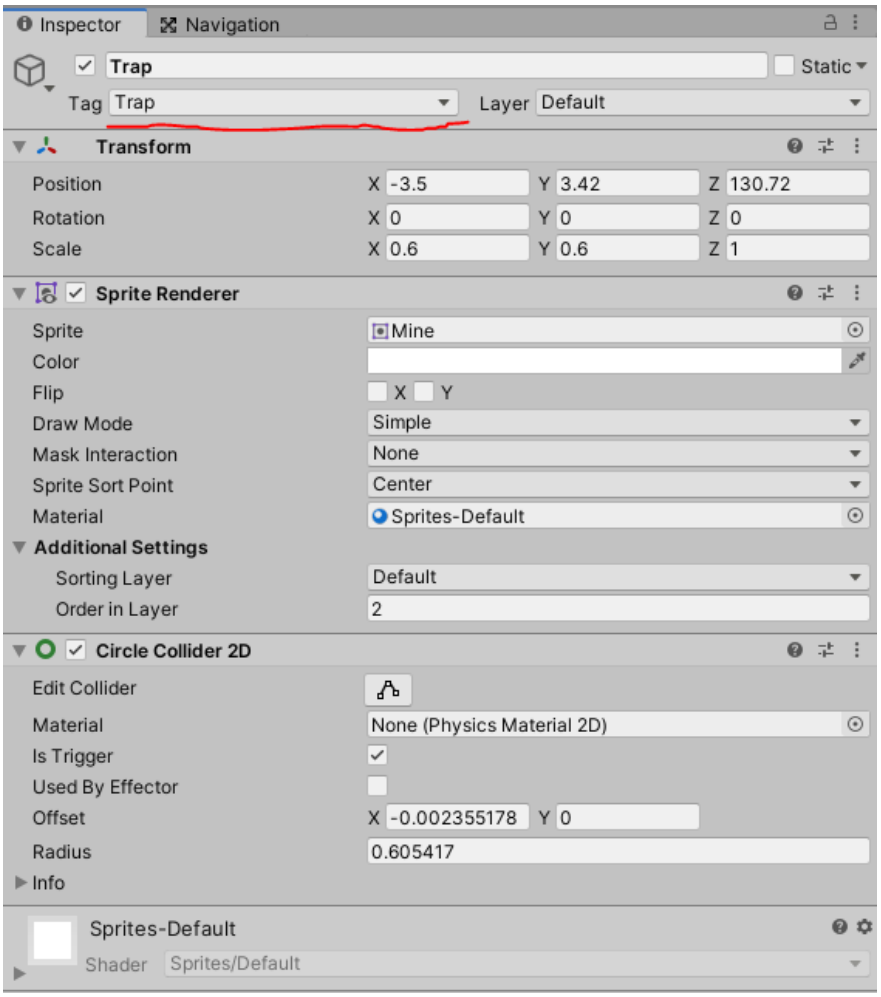
U Unity projekt igre dodano je 8 zvukova koji se koriste kroz igru. Svi zvukovi snimljeni su lupanjem predmeta ili korištenjem neartikuliranih ljudskih zvukova koji su potom uređivani u programu Audacity. U tablici 1 nalazi se lista zvukova i opis situacija u kojima se koriste.

Tablica 1: Lista i opis zvukova korištenih u igri

Zvuk	Trajanje	Opis
coin.mp3	<1 sec	Zvuk kada se pokupi novčić
gamelost.mp3	00:05	Zvuk koji se proizvede na kraju igre kada igrač izgubi
hunter-death.mp3	<1 sec	Zvuk proizveden po smrti lovca
hunter-swap.mp3	<1 sec	Zvuk koji se čuje kada dođe do promjene aktivnih lovaca
mine-drop.mp3	<1 sec	Zvuk koji se proizvede kada igrač postavi minu
mine-pickup.mp3	<1 sec	Zvuk skupljanja dijamanta
slime-vulnerable.mp3	00:01	Zvuk koji se proizvede kada zadnji lovac umre te ljigavac postane ranjiv
victory.mp3	00:03	Zvuk pobjede

3.4. Mina

Mina je kreirani objekt koji nije smješten nigdje ni na jednoj razini, ali zato u skripti Igrac.cs se nalazi „GameObject“ tip varijable koja se u skripti naziva „trapPrefab“ te se unutar Unitya proslijedi taj objekt kao parametar skripti. Potom kada igrač ima 1500 bodova ili više i pritisne razmaknicu stvori se instanca tog objekta i postavi na željenu lokaciju, u ovom slučaju na mjesto igrača. Mina kao objekt ima predefiniranu oznaku „Trap“ što je važno kada lovac dođe u kontakt sa minom jer se vrši provjera oznake objekta sa kojim je lovac došao u kontakt, te ukoliko se oznaka podudara tada se odvija cijela logika brisanja lovca. Ovako izgleda konfiguracija objekta mine:



Slika 13: Konfiguracija objekta mine (autorova slika 2021)

Te se ovom linijom koda mina instancira:

```
Instantiate(trapPrefab, player.position, player.rotation);
```

4. Kritički osvrt na igru i Unity

Unity je jedan od vodećih alata sa stvaranje igara na tržištu i nudi mnoštvo mogućnosti koje pripomažu kreiranju složenijih igara. Poprilično je intuitivan jednom kada se navikne na rad u njemu. Ponekad neke funkcionalnosti postanu naporne za konfigurirati ili jednostavno ne rade kako barem zvuči da bi trebalo kao što su sudarači.

U nekim trenutcima funkcije za rad sa sudarima koje se odnose na dugotrajniji kontakt ili prekid kontakta prilikom sudara ne rade kako bi trebale. Pretpostavljam da je to zbog toga što okidači/sudarači ne funkcioniraju najbolje sa algoritmom za traženje puta koji je dodan u ovaj projekt. No isto tako može biti da ne rade dobro sa skriptama krutog tijela. No pošto je algoritam za traženje puta nešto što originalno nije implementirano u Unity vjerojatno je to razlog za takvim problemima. Odnosno vizualno se sudari dogode, ali programska logika koja bi se trebala događati za trajanje sudara nekada ne radi kako spada.

Štoviše isti problem se odnosi i na okidače. Taj dio ovog alata predstavlja poteškoće pogotovo zato što su sudari i okidači veliki dio svake igre. No u ovoj igri su sudari i okidači uspješno konfigurirani. Igra je možda trebala sadržavati malo ljepšu grafiku odnosno dizajn likova i razina da je bilo na višoj dizajnerskoj razini. Igra koristi vanjsku biblioteku za navigiranje lovaca i ljigavca kroz labirint prepoznavajući prepreke. To je, primjerice, jedna od funkcionalnosti koju bih volio da Unity sadrži u sebi u obliku nekakve skripte.

5. Zaključak

Igra je osmišljena biti nalik popularnim naslovima „Pac-man“ i „Bomberman“ te utjelovljuje neke od karakteristika tih dviju opće poznatih igara, a uz to sadrži i osobni dodir, a što je: potreba brzog donošenja odluka. Tako da je ovo dosta brza igra koja zahtjeva mogućnost snalaženja po labirintu i reflekse.

Elementi igre rađeni su u mnogim alatima, no najviše u Unity alatu, te programska logika u Visual Studiu te su se ti programi iskazali kao prvoklasni alati za potrebe izrade igara. Jedini nedostatak kod Unity alata koji sam pronašao je nedostatak algoritama za traženje puta oko prepreka koju je bilo potrebno preuzeti sa interneta te ju implementirati u projekt. Druga stvar koja je malo nedostajala vezano za alate je način rada za traženje pogrešaka. Visual Studio ima odličan sustav traženja i označavanja pogrešaka dok Unity samo ispiše poruku o nastaloj greški i preusmjeri na skriptu u kojoj je greška nastala, a nije moguće skripte koje se koriste u Unity alatu testirati u Visual Studiu.

Ove se igra sastoji od 5 razina kroz koje igrač putuje labirintom i sakuplja novčiće koji mu daju bodove. Igrač osim novčićima dobiva bodove vremenski što je duže živ. Te bodove razmijeni za mine koje pronalazi u labirintu i postavlja iza sebe lovcima koji ga love. Kada lovac natrči na minu on se uništi. Eliminiranjem svih lovaca ljigavac koji se nasumično kreće po labirintu postaje ranjiv te kontakt sa igračem mu donosi smrt, a igraču pobjedu.

Igra sadržava mnogo potencijala za poboljšanjem i nadogradnjom. Jedna od nadogradnji koje bi bilo lijepo vidjeti je kompleksnija logika igre. Recimo da se aktivni lovci ubrzaju kada igrač nešto napravi itd. Svakako bi bilo lijepo vidjeti napredniju grafiku i dizajn labirinta koji stvara iluziju da je trodimenzionalan. Iako se igra čini završenom, kao da ima početak i kraj, mogu osmisliti još novih funkcionalnosti i kompliciranije mehanike. Primjer dviju od kompleksnijih mehanika su da ako igrač priđe na određeni domet ljigavcu da igrač postane sporiji ili da postoje mine koje protivnici ostavljaju. Tada bi se kretanje po labirintu igraču jako ograničavalo i morao bi jako paziti gdje se kreće i prema ćemu se kreće. Imam dojam da bih ovu igru mogao odlično unaprijediti kada bih nadograđivao u smjeru gdje lovci i ljigavac nisu jedina opasnost za igrača.

Popis literature

[1] Unity Technologies, *Unity Engine* (2021) [Na internetu]. Dostupno: https://unity.com/

[2] Unity Technologies (bez dat.) *The leading platform for creating interactive, real-time content.* Dostupno: https://unity.com/ [pristupano 08.06.2021].

[3] J. Craighead, J. Burke i R. Murphy, „The Unity engine comes with complete documentation with examples for its entire API. This is the biggest benefit of Unity and leads to increased productivity when compared to other engines...“, *Using the Unity Game Engine to Develop SARGE: A Case Study,* [Na internetu]. Dostupno na: https://www.researchgate.net/profile/Jeffrey-Craighead/publication/265284198_Using_the_Unity_Game_Engine_to_Develop_SARGE_A_Case_Study/links/55d33fdf08aec1b0429f32f4/Using-the-Unity-Game-Engine-to-Develop-SARGE-A-Case-Study.pdf [Pristupljeno: lipanj 2021.]

[4] Microsoft, *Visual Studio 2019* (2019.) [Na internetu] Dostupno: https://visualstudio.microsoft.com/vs/

[5] Unity Documentation, „Creating and using scripts“ (28.05.2021) [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html [Pristupano 8.6. 2021.].

[6] Audacity, *Audacity* (2021.) [Na internetu]. Dostupno: https://www.audacityteam.org/

[7] Brackeys (2021.) „Introduction to AUDIO in Unity“ [Na internetu]. Pristupano u ožujku 2021. Dostupno: https://www.youtube.com/watch?v=6OT43pvUyfY

[8] „Pathfinding in 2D“ (2021.) [Na internetu]. Dostupno: https://arongranberg.com/astar/docs/pathfinding-2d.php [Pristupano: 21. 3. 2021.]

Popis slika

Popis tablica